Final Project
Kalman Filtering of Position Data

Prepared by: Matt Rochford

EE 525-01

12/13/18

Dr. Jane Zhang

Part 1 - 2D Position Filtering

Approach:

The system was modeled as shown in the block diagram in Figure 1. It was tested using two separate inputs as well as with one singular input and the filter performed better with two separate inputs. The only difference in the derivation was that with one input the G matrix would be 4x1 but with two inputs it was 4x2. The power spectral density of each input was set as 1 (W_1=W_2=1).

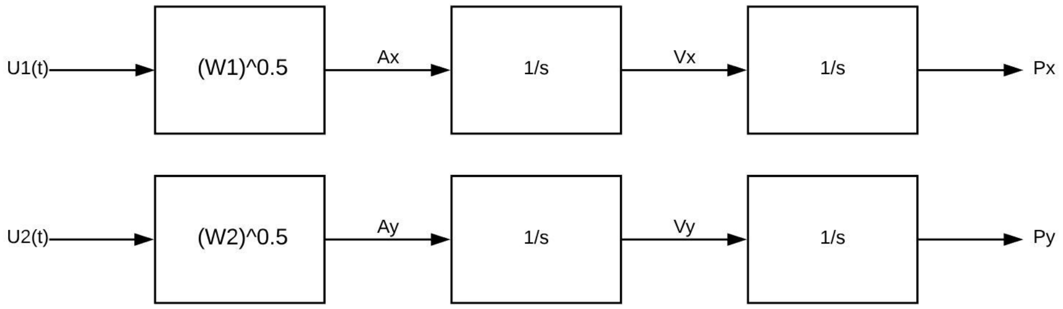

Figure 1 2D Block Diagram

The state equation is then derived from the block diagram and is shown below:

$$\begin{pmatrix} \dot{x_1} \\ \dot{x_2} \\ \dot{x_3} \\ \dot{x_4} \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} + \begin{pmatrix} 0 & 0 \\ \sqrt{W_1} & 0 \\ 0 & 0 \\ 0 & \sqrt{W_2} \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \end{pmatrix}$$

Using the F and G matrices, the rest of the discrete Kalman filter parameters are determined using the Van Loan method in Matlab. Technically, since the time step is not constant it would have been more accurate to analytically determine the phi and Q matrix in terms of delta t and recalculate the matrices each step through the filter. After using the Van Loan method first, the final results were good enough to not use the analytical method. This also saved computing time while running the filter since the matrices were only calculated once. The average of all time steps was used as the delta t value when calculating phi and Q. The variance of this time step data was low (about 0.2) further confirming the conclusion that using a single time step was a valid assumption.

The R matrix was set to be 100 times the identity matrix. This was chosen as the tunable parameter. The P minus was initialized as 0.1 times the identity matrix. The first position values from the dataset were used as the initial guesses for the X and Y position.

Results & Analysis:

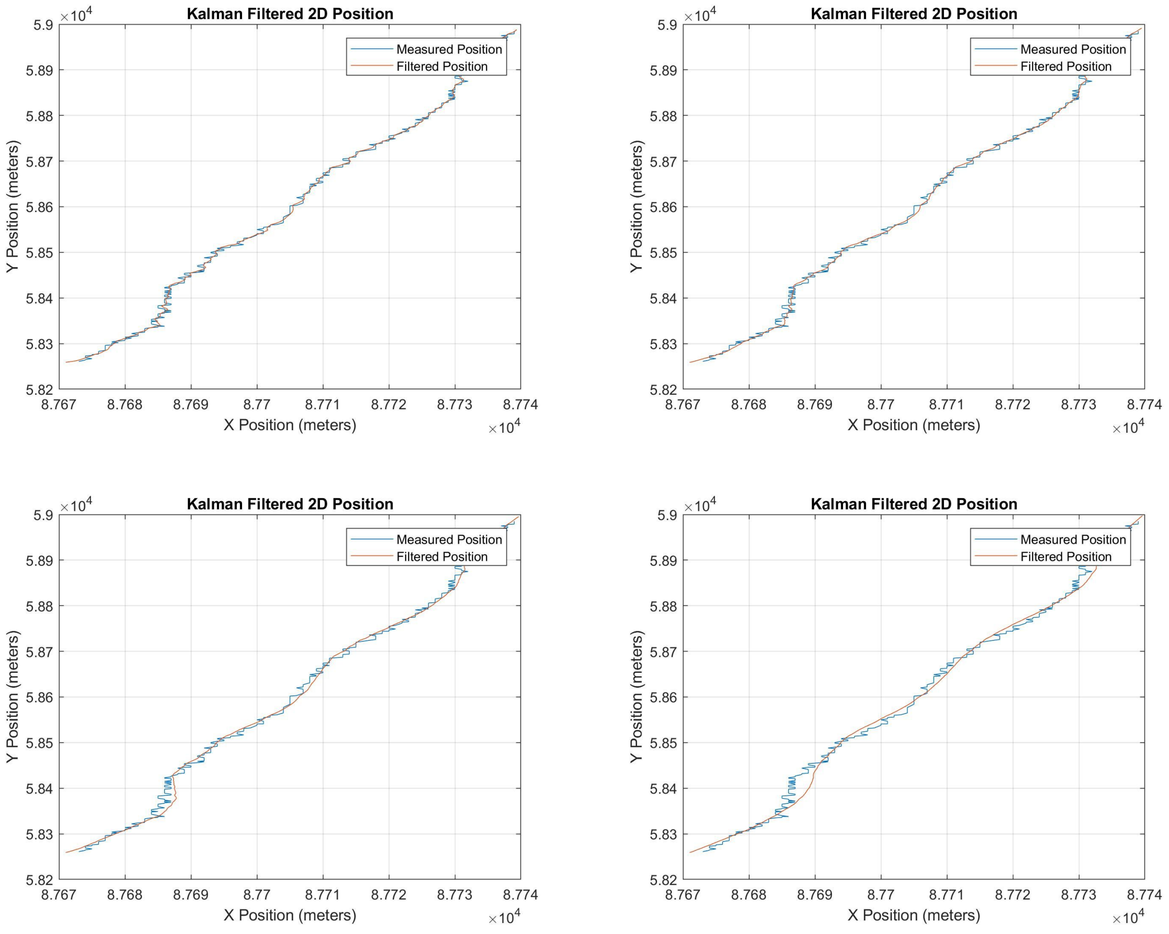

Figure 2 Kalman Filter outputs with R_K = 1,000 (top left), R_K = 10,000 (top right), R_K = 100,000 (bottom left), R_K = 1,000,000 (bottom right)

The measurement variance (R_K) was the value used as the tunable parameter. Changing the PSD of the input white noise had a similar effect as changing the R_K value and was tested. In order to account for an increase in PSD, a similar increase had to be applied to R_K to prevent the filter outfit from overfitting to the measurement. From figure 2 it can be seen that R = 1,000,000 was too high of a variance due to the deviation from the position path. It is hard to make a definitive statement about the best setting without knowing the true path of the vehicle but an R value in between 100,000 and 10,000 would be suggested from this experiment.

Another tuning possibility would be to adjust the PSD of each white noise to be different. For this example the movement in each position was relatively similar so the tuning here would be minimal but could possibly improve results further. All code for part 1 is included in Appendix I.

Part 2 - 3D Position Filtering

Approach

The program from part 1 needed minor modification to work with 3D position data. Since the Van Loan method was set up based on the dimension of the X vector (n) the program was portable to an increase in dimensions. The 3D system was modeled in the block diagram shown below by adding a third white noise input. This changed the F matrix to be 6x6 and the G matrix to be 6x3.

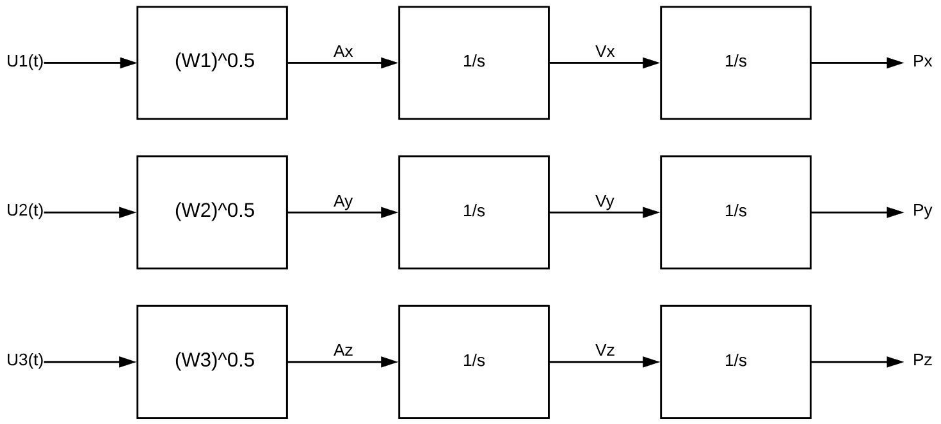

Figure 3 3D Block Diagram

The state equation is then derived from the block diagram and is shown below:

$$\begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \\ \dot{x}_5 \\ \dot{x}_6 \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{pmatrix} + \begin{pmatrix} 0 & 0 & 0 \\ \sqrt{W_1} & 0 & 0 \\ 0 & 0 & 0 \\ 0 & \sqrt{W_2} & 0 \\ 0 & 0 & 0 \\ 0 & 0 & \sqrt{W_3} \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix}$$

Results & Analysis:

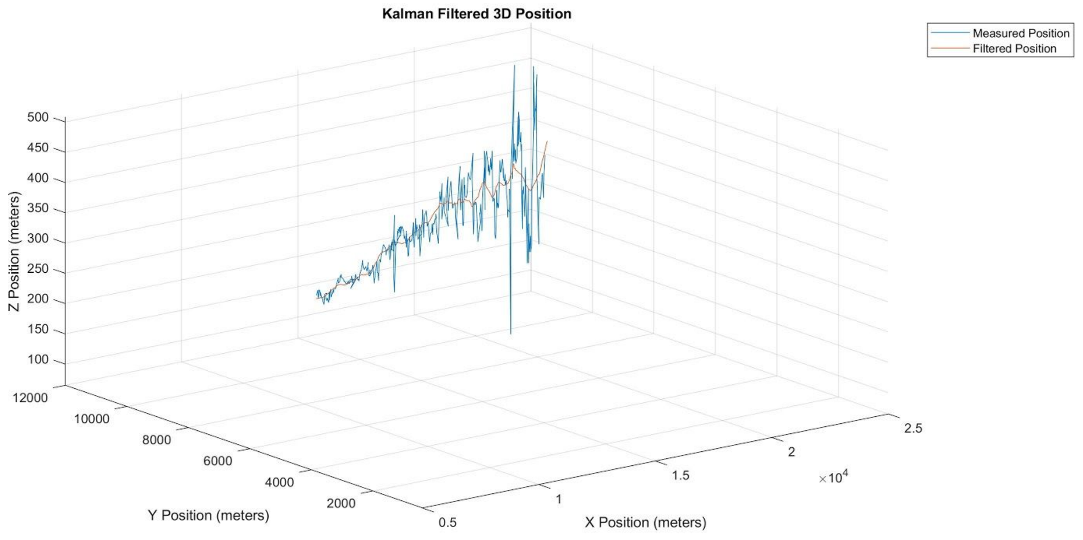

Figure 4 3D Position Filtering with R = 10,000

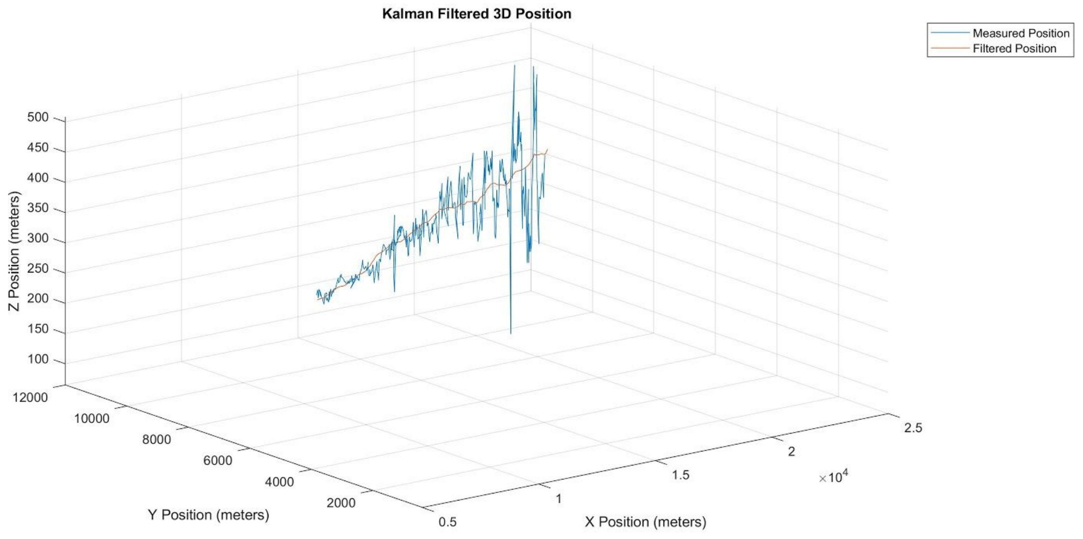

Figure 5 3D Position Filtering with R = 100,000

Figure 4 and figure 5 are the two best results from the 3D filtering. Figure 5 has too general of a response whereas figure 4 seems closer to the true position. Once again it is hard to definitively say the best value without knowing true position but a value close to R = 10,000 would be suggested by this experiment.

Appendix I

Sript for Part 1 (2D)

```
%% EE 525 Final Project
% Kalman Filter for 2D Position Tracking

clc
clear all
close all

%% Import Dataset
% Important position data
fileID = fopen('2Ddata_r.txt','r');
formatSpec = '%*s %d %d';
sizeA = [2 Inf];
A_pos = fscanf(fileID,formatSpec,sizeA); % Position values in meters

% Import file as single string
fileID = fopen('2Ddata_r.txt','r');
formatSpec = '%c';
sizeA = [1 Inf];
A_time = fscanf(fileID,formatSpec,sizeA);

% Initialize values
colon = 0;
i = 1;
j = 1;

% Extract time data from string
while i < length(A_time)
    if (A_time(i) == ':')
        colon = colon + 1; % Increment colon flag
    end
    if colon == 2
        colon = 0; % Reset colon flag
        time(j) = str2num(A_time(i+1:i+6)); % Store relevant data
        j = j + 1;
    end
    i = i + 1;
end

% Convert time data to dt values
for i = 1:length(time)-1
    dt(i) = time(i+1)-time(i);
end
% Correct negative values
dt(dt<0) = dt(dt<0)+60;

% Calculate velocities
%for i = 1:length(dt)
%    A_vel(1,i) = (A_pos(1,i+1)-A_pos(1,i))/dt(i); % Y velocity
%    A_vel(2,i) = (A_pos(2,i+1)-A_pos(2,i))/dt(i); % X velocity
%end

% Assign final measurement values
len = length(A_pos);
x0 = A_pos(2,1);
y0 = A_pos(1,1);
```

```
y_pos = A_pos(1,2:len);
x_pos = A_pos(2,2:len);
%y_vel = A_vel(1,:);
%x_vel = A_vel(2,:);

% Clear unneeded variables
clearvars -except y_pos x_pos x0 y0 dt

%% Design state model
W1 = 1; % PSD for X acceleration
W2 = 1; % PSD for Y acceleration

F = zeros(4);
F(1,2) = 1;
F(3,4) = 1;

G = zeros(4,2);
G(2,1) = sqrt(W1);
G(4,2) = sqrt(W2);

H = [1 0 0 0; 0 0 1 0];

%% Initialize Phi and Q
sigma = var(dt);
dt = mean(dt); % Sampling interval
n = 4; % Dimension of x(t)

% Compute A matrix
A = [-F*dt G*W1*G'*dt; zeros(n) F'*dt];

% Compute B matrix
B = expm(A);

% Compute Phi and Q
phi = B(n+1:2*n,n+1:2*n)';
Q = phi*B(1:n,n+1:2*n);

% Initialize Rk (measurement variance)
R = 10000*eye(n/2);

%% Implement Kalman Filter

P_ = 0.1*eye(n); % Initialize error
X_ = [x0; 0; y0; 0]; % Initial guess
Z = [x_pos; y_pos];

for k = 1:length(x_pos)
    % Measurement Update
    K = P_*H'*inv(H*P_*H'+R); % Compute Kalman Gain
    X(:,k) = X_+K*(Z(:,k)-H*X_); % Update estimate
    P = (eye(n)-K*H)*P_;

    % Time Update
    X_ = phi*X(:,k); % Project state ahead
    P_ = phi*P*phi'+Q; % Project error ahead

end

figure
plot(x_pos,y_pos,X(1,:),X(3,:))
```

```
legend('Measured Position','Filtered Position')
title('Kalman Filtered 2D Position')
xlabel('X Position (meters)')
ylabel('Y Position (meters)')
grid on
```

Appendix II

Script for Part 2 (3D)

```
%% EE 525 Final Project
% Kalman Filter for 3D Position Tracking

clc
clear all
close all

%% Import Dataset
% Important position data
fileID = fopen('3Ddata_r.txt','r');
formatSpec = '%*s %f %f %f';
sizeA = [3 Inf];
A_pos = fscanf(fileID,formatSpec,sizeA); % Position values in meters

% Import file as single string
fileID = fopen('3Ddata_r.txt','r');
formatSpec = '%c';
sizeA = [1 Inf];
A_time = fscanf(fileID,formatSpec,sizeA);

% Initialize values
colon = 0;
i = 1;
j = 1;

% Extract time data from string
while i < length(A_time)
    if (A_time(i) == ':')
        colon = colon + 1; % Increment colon flag
    end
    if colon == 2
        colon = 0; % Reset colon flag
        time(j) = str2num(A_time(i+1:i+6)); % Store relevant data
        j = j + 1;
    end
    i = i + 1;
end

% Convert time data to dt values
for i = 1:length(time)-1
    dt(i) = time(i+1)-time(i);
end
% Correct negative values
dt(dt<0) = dt(dt<0)+60;

% Assign final measurement values
```

```
len = length(A_pos);
x0 = A_pos(1,1);
y0 = A_pos(2,1);
z0 = A_pos(3,1);
x_pos = A_pos(1,2:len);
y_pos = A_pos(2,2:len);
z_pos = A_pos(3,2:len);

% Clear unneeded variables
clearvars -except y_pos x_pos z_pos z0 x0 y0 dt

%% Design state model
W1 = 1; % PSD for X acceleration
W2 = 1; % PSD for Y acceleration
W3 = 1; % PSD for Z acceleration

F = zeros(6);
F(1,2) = 1;
F(3,4) = 1;
F(5,6) = 1;

G = zeros(6,3);
G(2,1) = sqrt(W1);
G(4,2) = sqrt(W2);
G(6,3) = sqrt(W3);

H = [1 0 0 0 0 0; 0 0 1 0 0 0; 0 0 0 0 1 0];

%% Initialize Phi and Q
sigma = var(dt);
dt = mean(dt); % Sampling interval
n = 6; % Dimension of x(t)

% Compute A matrix
A = [-F*dt G*W1*G'*dt; zeros(n) F'*dt];

% Compute B matrix
B = expm(A);

% Compute Phi and Q
phi = B(n+1:2*n,n+1:2*n)';
Q = phi*B(1:n,n+1:2*n);

% Initialize Rk (measurement variance)
R = 10000*eye(n/2);

%% Implement Kalman Filter

P_ = 0.1*eye(n); % Initialize error
X_ = [x0; 0; y0; 0; z0; 0]; % Initial guess
Z = [x_pos; y_pos; z_pos];

for k = 1:length(x_pos)
    % Measurement Update
    K = P_*H'*inv(H*P_*H'+R); % Compute Kalman Gain
    X(:,k) = X_+K*(Z(:,k)-H*X_); % Update estimate
    P = (eye(n)-K*H)*P_;

    % Time Update
    X_ = phi*X(:,k); % Project state ahead
    P_ = phi*P*phi'+Q; % Project error ahead

end

figure
plot3(x_pos,y_pos,z_pos,X(1,:),X(3,:),X(5,:))
legend('Measured Position','Filtered Position')
title('Kalman Filtered 3D Position')
xlabel('X Position (meters)')
ylabel('Y Position (meters)')
zlabel('Z Position (meters)')
axis([5000 25000 380 12000 min(z_pos) max(z_pos)])
grid on
```